AFRL-RI-RS-TR-2015-132

GRAPHTABLES

JUNE 2015

INTERIM TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2015-132 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
ASHER D. SINCLAIR
Work Unit Manager

/ S /
JULIE BRICHACEK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUNE 2015	FINAL TECHNICAL REPORT	OCT 2013 – SEP 2014

4. TITLE AND SUBTITLE

GRAPHTABLES

5a. CONTRACT NUMBER
IN-HOUSE: R12L

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Jason A. Moore

5d. PROJECT NUMBER
PAVZ

5e. TASK NUMBER
IH

5f. WORK UNIT NUMBER
07

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RISB
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2015-132

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2015-2638
Date Cleared: 26 MAY 2015

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Graphtables provides for a spreadsheet style structured exploration, filtering, and augmenting of graphs. This document introduces the Graphtables concept and compares it to current methods and issues that common methods present. A subset of graph specific tasks from published works is then listed and how the Graphtables metaphor could directly enhance many of the tasks that generalized graph visualizations fail to provide.

15. SUBJECT TERMS
Graph Visualization, Graph Navigation, Visual Literacy

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
U	U	U	UU	37	**DAVID E. KAVENEY**
					19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Table of Contents

List of Figures

List of Tables

1.0 Summary

Graphtables provides for a spreadsheet style structured exploration, filtering, and augmenting of graphs. This document introduces the Graphtables concept and compares it to current methods and issues that common methods present. A subset of graph specific tasks from published works is then listed and how the Graphtables metaphor could directly enhance many of the tasks that generalized graph visualizations fail to provide.

2.0 Introduction

User driven path traversal on graphs is cognitively difficult and error prone. The problem of exploring graphs becomes impossible when the nodes and links of graphs become larger than available display real-estate, making it impossible for the viewer to follow the embedded relations. A recent survey paper of graph visualization techniques (Gibson, Faith, & Vickers, 2012), cites the difficulty of common graph tasks (identifying the shortest path, determining connectivity, listing neighbors, etc.) in even small graphs of 10s of nodes. Additionally, they discuss that cognitively inspired layouts aiming to reduce edge crossing or otherwise making improvements to graph aesthetics also failed to improve the user's performance.

2.1. Traditional node-link

The following images showcase the most common way of displaying graphs, as a series of links and nodes.

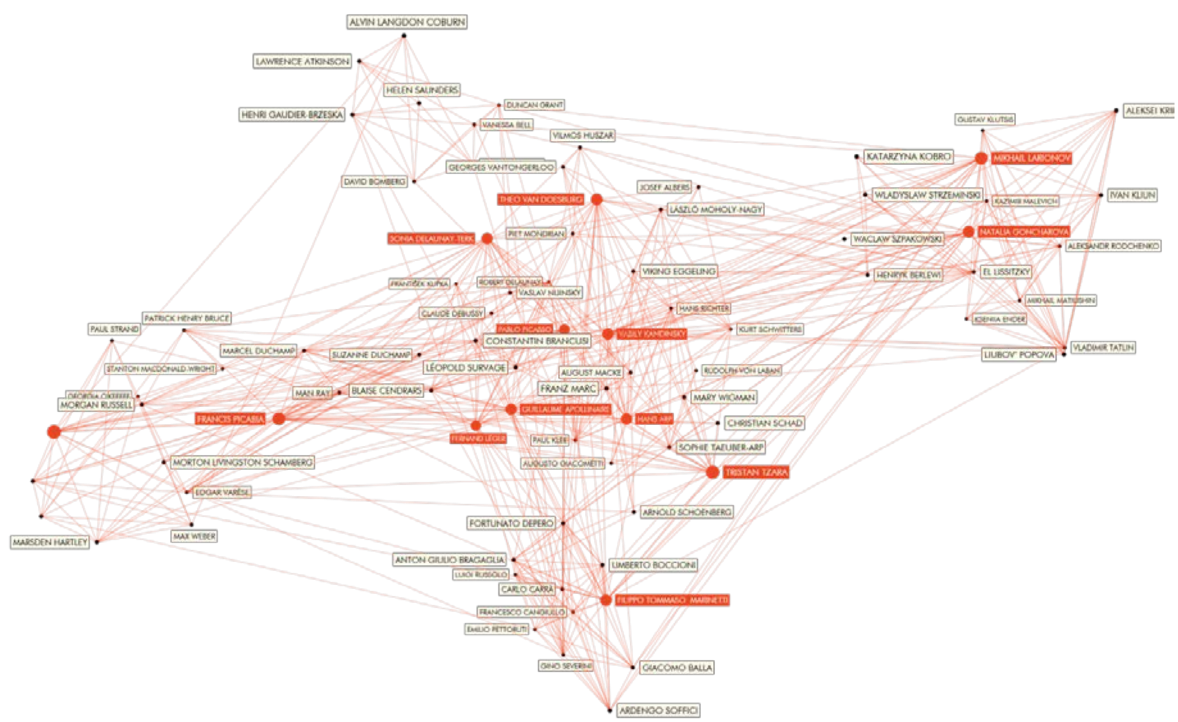


Figure 1 ***Inventing Abstraction*** **1910-1925 Graph of artists and their connections with other artists that where pivotal to the field of abstract art. ((MOMA), 2012)**

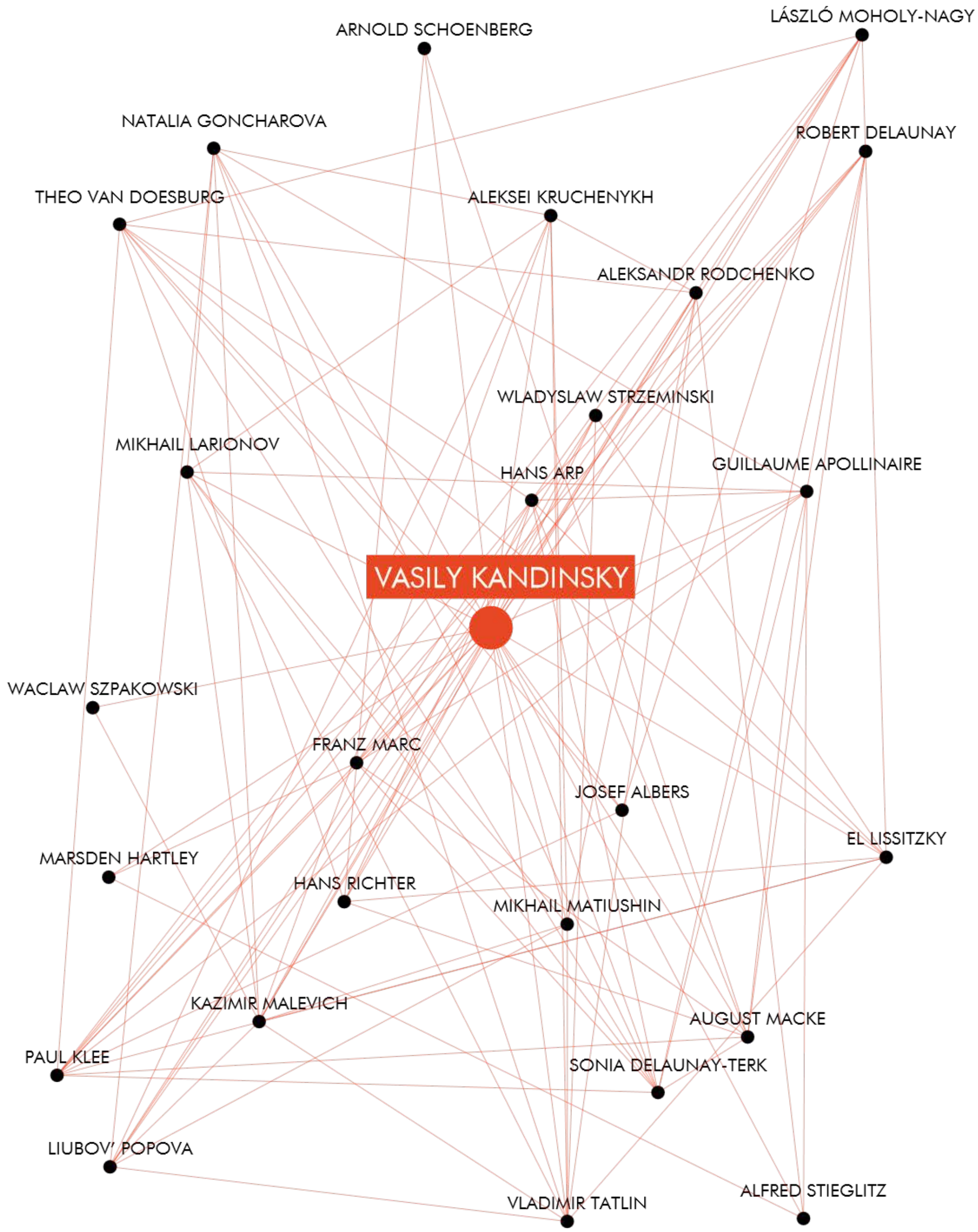


Figure 2 The effect of selecting Vasily Kadinsky from the overview graph displays the artist's direct connections. ((MOMA), 2012)

While the graph from Figure 1 is greatly simplified when selecting a particular artist, as displayed in Figure 2, difficulty in following links between participants is exacerbated by numerous overlapping edges, obscuring labels, and a layout that attempts to show the important figure centrally. Imagine using this depiction to determine if there is a direct connection between Mikhal Matiushin and Natalia Goncharova.

2.2. Large Graphs

Many techniques for making large graphs more aesthetically pleasing make discerning detail in the graph more difficult where they allege the opposite. Perhaps the best example of this is in Force-Directed Edge Bundling.

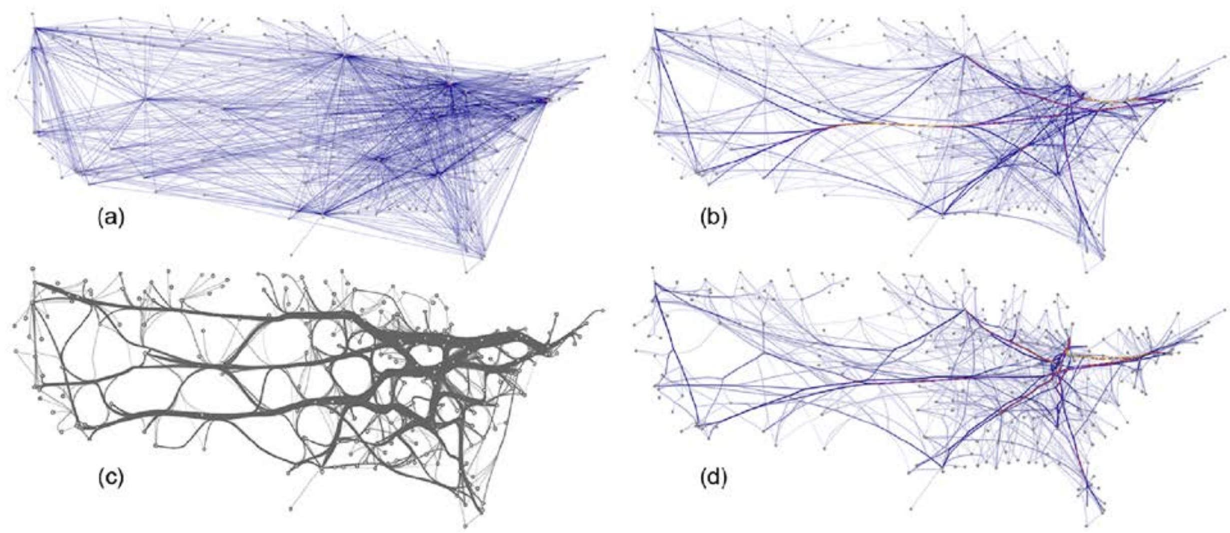


Figure 3 US airlines graph (235 nodes, 2101 edges) (a) not bundled and bundled using (b) [Force Directed Edge Bundling] FDEB with inverse-linear model, (c) [Geometry-Based Edge Bundling] GBEB and (d) FDEB with inverse-quadratic model. (Holten & Van Wijk, 2009)

While the (b), (c), and (d) figures above are indeed more visually appealing, the details are heavily obscured and individual edges that could be traversed before bundled are now completely lost among the bundled edges.

2.3. Visual Literacy

Visual literacy is the most dominant factor in a person being able to use information embedded into any visual metaphor. Without it, the user requires a significant amount of effort in first understanding how the visual representation is meant to convey information. Results become worse when the user misinterprets the visualization and applies the incorrect set of assumptions which can lead to incorrect conclusions. Key journals, such as Nature, expect users to have a robust visual literacy and have articles online to help readers understand their means and methods. These depictions are designed to help scientists and laypeople make sense of the complicated world in which we live. The following images are excerpts from nature|methods[1], a website dedicated to teaching techniques for life scientists and chemists.

[1] http://www.nature.com/nmeth/index.html

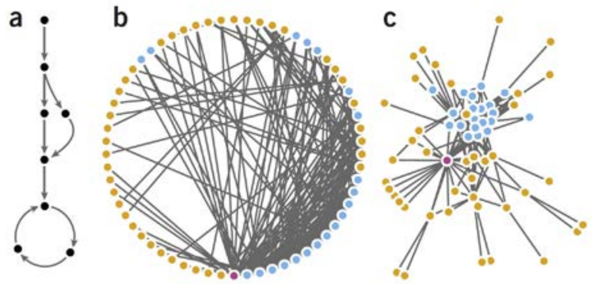


Figure 4 Node-link diagrams. (a) A directed graph typical of a biological pathway (b) An undirected graph with nodes arranged in a circle. (c) A spring-embedded layout of data from b. (Nature America Inc, 2012)

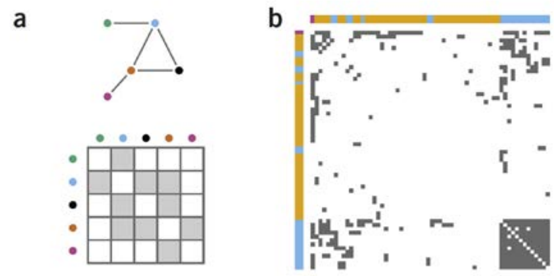


Figure 5 Adjacency matrices. (a) Nodes are ordered as rows and columns; connections are indicated as filled cells. (b) A matrix representation of data from Figure [4]. (Nature America Inc, 2012)

For a more complete understanding of the breadth of graph visualization techniques, the reader is encouraged to read, "A survey of two-dimensional graph layout techniques for information visualization" (Gibson, Faith, & Vickers, 2012).

2.4. Tasks

Traditional graph visualization designs are not designed to directly address a user's questions and therefore attempt to optimize on visual aesthetics. Understanding the tasks (types of questions) that users are trying to answer with graphs should provide substantial pressure in designing optimal graph visualizations. The tasks as defined by (Lee, Plaisant, Parr, Fekete, & Henry, 2006) describe primitive task operations; the intent of Graphtables is to improve user performance in many of these tasks. While Graphtables is about optimizing human performance, no formal human experiment was performed under this task; however, many of the computer related issues that are of concern are introduced below. The Graphtables concept and basic implementation[2] is patent pending: PASN 62/087,289, titled, "METHOD AND APPARATUS FOR GRAPHICAL DATA INTERACTION AND VISUALIZATION OF GRAPHS VIA PATHS".

3.0 Methods, Assumptions, and Procedures

Initial exploration of the Graphtables approach was expressed as a series of concept art with an evolving set of visual options and a set of notes and discussion points. Relevant portions of that exploration are in the Results and Discussion sections below. Once that was accomplished, a small curated dataset (4 nodes and 2 edges) was created to test and illustrate a subset of features. Once the tests were created and the output was validated, a very primitive interactive visualization was implemented which allowed the user to perform a simple sort and led to patenting the concept.

The overall assumption of Graphtables is simple; structured navigation of graphs will outperform commonly used layout algorithms for a large range of graph specific tasks. Those tasks are enumerated from previous published works by Lee et al and fall into one of the following: topology-based tasks, attribute-based tasks, browsing tasks, and overview tasks.

[2] AFRL/RHCV has volunteered resources to assist in developing a full-featured Graphtables application.

4.0 Results and Discussion

Graph visualization algorithms are conceptually designed to support human decision making. As graphs of larger sizes are used, they often fall short in providing direct answers and only provide some overview capability. Instead of designing a general graph visualization capability which is agnostic to the types of questions users need (or want) answered, Graphtables uses a tabular metaphor that allows the user to directly answer graph oriented questions.

Lee et al define four major types of tasks: topology-based, attribute-based, browsing and overview. Topology-based tasks are broken down into: adjacency (direct connection), accessibility (direct or indirect connection), common connection, and connectivity. Attribute tasks are broken down into: on the nodes, and on the links. Browsing tasks are broken down into: path following and revisiting. Overview visualization tasks as defined by Lee et al are not treated in this work and it is believed that Graphtables will perform worse since its purpose is specifically to show individual paths through the graph and not the graph structure in its entirety.

The following is an excerpt from (Lee, Plaisant, Parr, Fekete, & Henry, 2006) where they describe one type of topological task and introduce some example questions:

> Find the set of nodes adjacent to a node.
> · How many nodes are adjacent to a node?
> · Which node has a maximum number of adjacent nodes?
>
> Examples:
> · Find the names of the direct friends of Eric.
> [Find on Nodes + Find Adjacent Nodes on Nodes + Retrieve Value on Nodes]
> · How many kinds of organisms do golden eagles eat?
> [Find on Nodes + Find Adjacent Nodes on Nodes + Filter on Links + Compute Derived Value (Count) on Nodes]
> · Who is the most popular person?
> [Find Extremum on Nodes]

The above tasks are not well supported by traditional graph visualizations as exemplified in Figure 1. Take for instance, the problem of finding the neighbors of Vasily Kandinsky. This first requires the user to scan the entire graph for the name [*Find on Nodes*], or alternatively, to use a search function if it is available in the tool. Unfortunately, due to the number of edges in the graph, it is nearly impossible to walk the edges. Instead, the user would have to prune the graph to see only the direct neighbors as can be seen in Figure 2. At this point, the user could answer the question of retrieving the names of the neighbors of Vasily Kandinsky. However, finding all the second degree neighbors is significantly harder and error prone since the relevant data for each task would require the user to repeat the above tasks for each of the direct neighbors.

These tasks become nearly impossible when the query parameter doesn't fit well into the interface, such as attributes or names that do not reasonably fit within the constrained area. It is rare for graph

visualizations to encode secondary attributes into the visualization; if they are represented, they are typically encoded as shapes or colors of the nodes and edges. This dramatically reduces the number of values that can be displayed, reducing the amount of information available to the viewer.

4.1. Graphtables Introduction

Graphtables provides the user with a spreadsheet style visual and interaction metaphor to sort, augment, and filter paths over arbitrary graphs. Take the following graph G(v,e) = ({A,B,C,D}, {{A,C}, {A,D}}) where each of the nodes and edges also have the following attributes:

Table 1 Simple graph of nodes, edges, and attributes.

Nodes	Label	Value	Valid
A	A	3	T
B	B	NULL	F
C	C	2	F
D	D	9	NULL

Edges	Label	Weight
AC	AC	10
AD	AD	2

The graph can be depicted in the following traditional way:

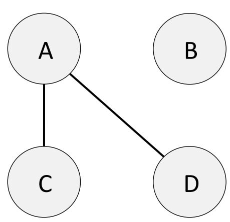


Figure 6 Simple graph depicted as nodes and edges.

Instead of displaying the graph in the traditional way, Graphtables first builds a complete set of paths starting from every node and of every length possible such that each path contains no cycles and no path is represented more than once. Each path is then displayed in a single row with each node visually represented by an encapsulated rounded rectangle and each edge as a line.

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T										
B		F										
C	2	F										
D	9											
A	3	T	AC	10	C	2	F					
C	2	F	AC	10	A	3	T					
A	3	T	AD	2	D	9						
D	9		AD	2	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
D	9		AD	2	A	3	T	AC	10	C	2	F

Figure 7 Graphtables depiction of a simple graph.

The path metaphor of alternating between vertices and edges is additionally encoded in the main header which alternates between Node and Edge until the length of the longest path is reached. Below those main headers, sub-column headers are provided for each attribute which exists within the graph.

The operations then available to the user are relatively simple: Sort by any number of sub-columns, insert a main level column, insert a sub-column, change the order of the sub-columns, filter the graph, filter a main column, or filter by a sub column.

Displaying all possible paths does grow quickly and will be very large for real-world graphs. The following formula provides the upper bound of this complexity:

$v + 2 \cdot \binom{n}{2} \cdot \sum_{i=0}^{n-2} \left(i! \cdot \binom{n-2}{i} \right)$ where $n = \left\lceil \frac{1+\sqrt{1+8e}}{2} \right\rceil$ where ***v*** is the total number of vertices in the graph and ***e*** is the total number of edges[3]. The explosion of real-estate needed to display these paths is not considered a detriment and during real-world tasks is believed to be of great utility and is discussed in later sections.

4.2. Sort by sub-columns

One of the primary operations provided to the users is the ability to sort one or more columns. Figure 8 displays the result of taking the simple graph and sorting the paths by the values in Node1's sub-column.

From this, it is then trivial for the user to answer questions about which nodes have the greatest values, least values, or the distribution of values of any attribute.

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value ▼	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
D	9											
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F
A	3	T										
A	3	T	AC	10	C	2	F					
A	3	T	AD	2	D	9						
C	2	F										
C	2	F	AC	10	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
B		F										

Figure 8 Graphtables sorting by sub-column.

As is possible with spreadsheet programs, the user can cascade sorting parameters. Figure 9 depicts the result of the user performing a secondary sort on Edge 1's Weight sub-column.

[3] Special thanks to Dr. Victoria Horan of AFRL for her assistance in the derivation of this upper bound.

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value▼	Valid	Label	Weight▲	Label	Value	Valid	Label	Weight	Label	Value	Valid
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F
D	9											
A	3	T	AD	2	D	9						
A	3	T	AC	10	C	2	F					
A	3	T										
C	2	F	AC	10	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
C	2	F										
B		F										

Figure 9 Graphtables secondary sort.

It is reasonable to expect that graphs with heterogeneous attributes will be ingested. In the current version, any node or edge that does not have a value for that sub-column will results in a visually empty cell. This has an interesting implication; the number of sub-columns for each of the nodes and edges is the sum of unique attribute types between all nodes and edges, respectively.

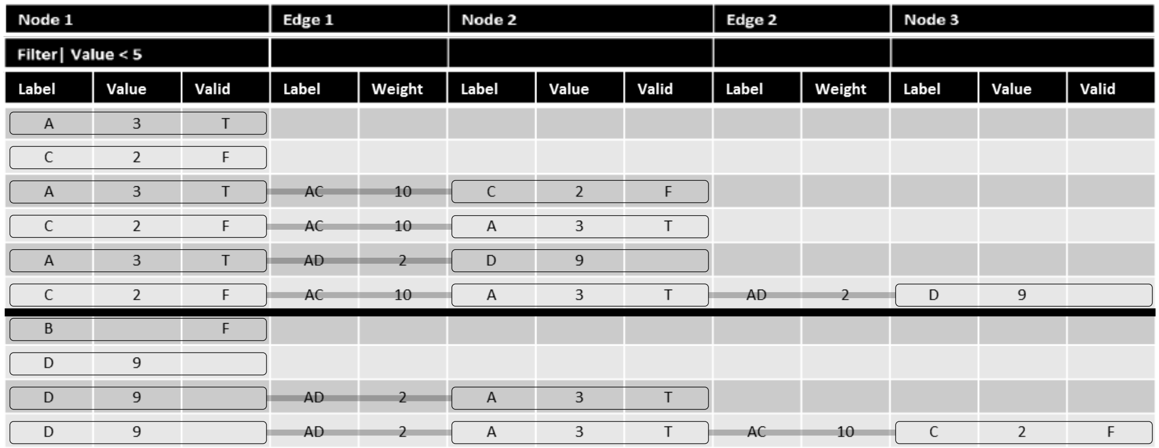
4.3. Filtering

Another common task will be applying one or more filters to the data. Filtering can be done on the entire graph, on a particular Node or Edge main column, or on any sub-column. Figure 10 depicts one possible way to display a whole graph filter. The user provides a filter predicate and the system then sorts the paths to reflect that portions of the graph either pass or fail the filter. Notice that this filtering is done prior to any other user defined sorting as defined in section 4.2.

Filter| Value < 5												
Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T										
C	2	F										
A	3	T	AC	10	C	2	F					
C	2	F	AC	10	A	3	T					
B		F										
D	9											
A	3	T	AD	2	D	9						
D	9		AD	2	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
D	9		AD	2	A	3	T	AC	10	C	2	F

Figure 10 Graphtables example of a whole graph filter (Filter on Value < 5 as displayed in the top bar).

This essentially converts the filtering operation into a ranking operation, meaning that paths that fail the filter are still ranked with respect to the user defined sorting. This is vastly different from every other known graph representation. To put it another way, filtering is just ranking and a user will never be presented an empty set. Instead, the paths that most closely satisfy the desires of the user are presented earlier in the list. This is keeping with the intent of preserving the Gestalt principle of visualization where items that are closer are more related than items which are further away.



Node 1			Edge 1		Node 2			Edge 2		Node 3		
Filter| Value < 5												
Label	Value	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T										
C	2	F										
A	3	T	AC	10	C	2	F					
C	2	F	AC	10	A	3	T					
A	3	T	AD	2	D	9						
C	2	F	AC	10	A	3	T	AD	2	D	9	
B		F										
D	9											
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F

Figure 11 Graphtable example of a sub-column filter.

As described above, filters can be applied to sub-portions of the graph and Figure 11 depicts the result of the same basic filter to a sub-column. Notice the partition line is between rows 6 and 7 versus when it was applied to the whole graph filter and the partition was between rows 4 and 5.

How filtering should be specified by the user is still open. A simple and natural mechanism can be something as simple as a Boolean algebra that allows the operator to choose a column and then an inequality and value for comparison. Limiting the interface to Boolean operations is not necessary and more expressive interfaces can enable extrinsic data from the graph to be represented. One could imagine specifying a geospatial filter defined through a map interface to filter out nodes that have addresses further from some user generated polygon. Alternatively, the user could create a custom column that calculates distance information and then perform a simple sort and filter. Even more expressive mechanisms for sorting are possible and could include domain specific languages or other graphical techniques for specifying the filter criteria.

4.4. Edge Bundling

A fair criticism of the approach as presented thus far is that paths that have many common elements require the user to visually match each of the elements to identify the common subgraph. While the default ordering will result in rows having many common sub-paths, direct neighbors will have much less in common once user defined filtering and sorting is applied. Techniques similar to edge bundling can still be supported, and unlike the common case where the edge bundling technique makes it impossible to follow exact paths, it enhances the readability in the Graphtables implementation. Take Figure 12 for instance. Neighboring duplicate nodes and edges are collapsed, resulting in an easier to discern set of common sub paths. There are numerous other ways that edge bundling could be visually depicted, and for example, Figure 13 shows an alternative where there are no repeated nodes or edges in subsequent columns. One issue with this representation is that sub paths (for example the path that starts and end with the C labelled Node) are not as visually salient as in the other representations. Figure 14 represents one possible compromise between drawing the minimum number of repeated values and drawing each path independently.

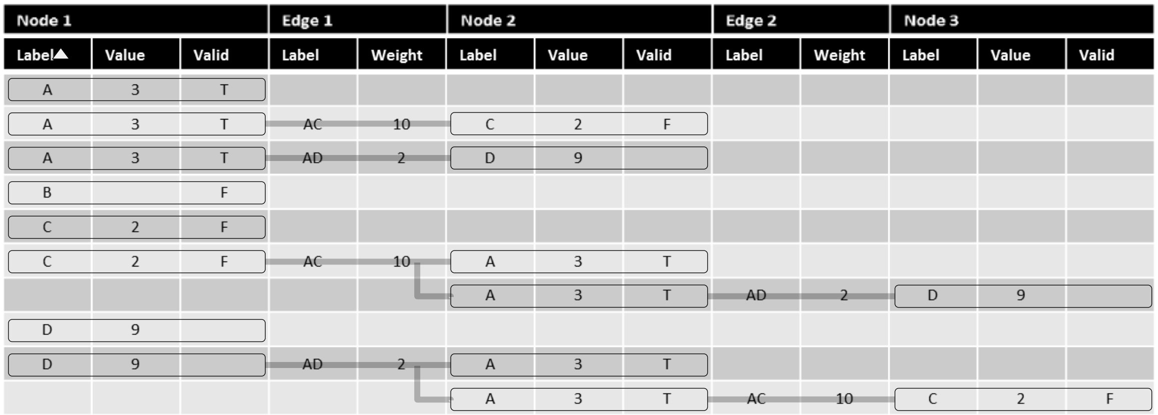


Figure 12 Edge-bundling common neighbor sub paths.

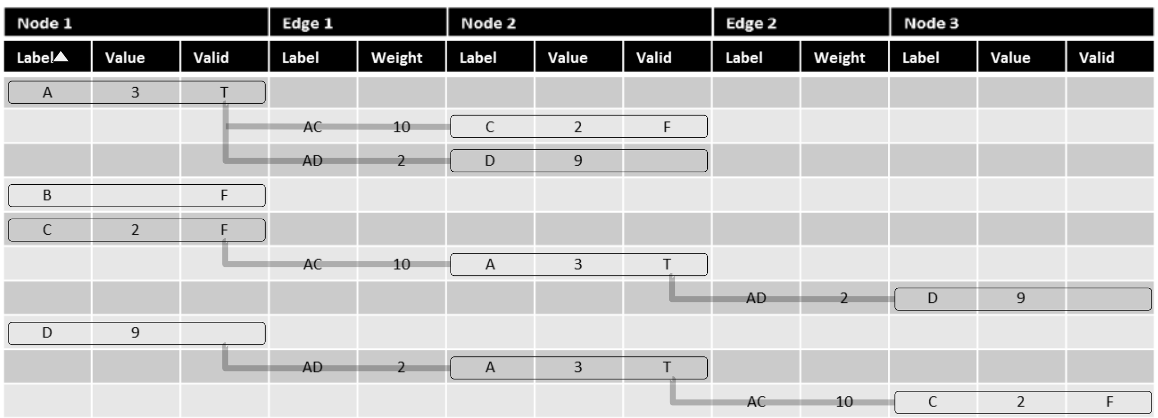


Figure 13 Edge-bundling all repeated nodes and edges.

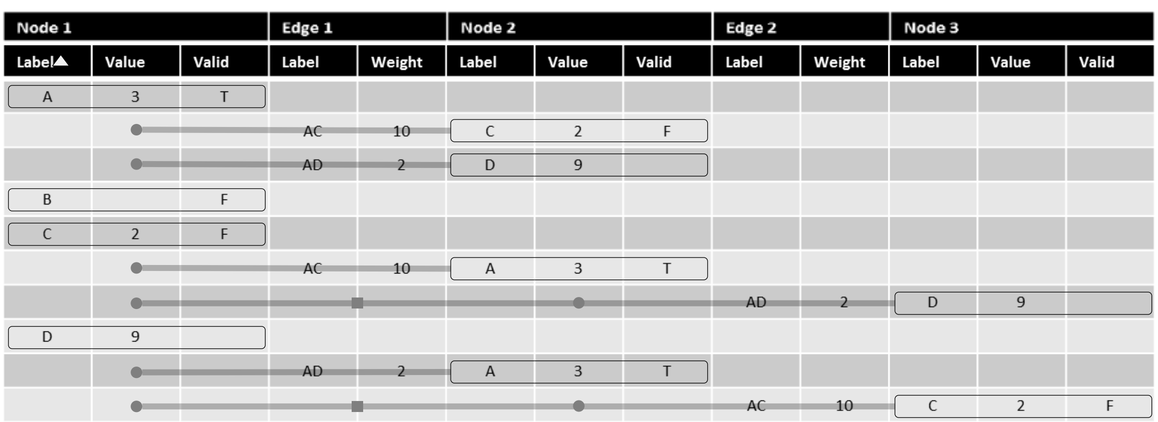


Figure 14 Edge-bundling compromise.

4.5. Custom Columns

Adding user defined columns, both at the main level and sub-level is very desirable. This feature allows users to augment data with values that are either necessary to answer questions intrinsic within the graph or calculated values such as path length. More interesting is augmenting the graph with values that are not internally available, like distance of addresses in the graph from a user specified location. While not depicted, the expectation is that users could move these columns freely at the main level if created there, or within the sub columns if created under a particular main level header. It may also be

desirable when users insert a column under one node that it is replicated under all nodes, but these interface concepts need to be explored for good defaults and allow for alternative implementations.

4.6. Secondary features

There are a set of secondary features that will make using larger graphs or graphs with numerous attributes easier to handle. Future implementations should provide the ability to hide undesired or unnecessary attribute columns for either a particular main column or for the entire display. Additionally, the user could also decide to hide rows from the display. Both of these features are common in high quality spreadsheet software and should be intuitive for users to understand.

Altering the sorting precedence in spreadsheets is difficult and often requires the user to completely re-define the sorting preferences. Some effort providing a more intuitive method is necessary since Graphtables is designed for these sorting properties to be altered often and knowing the order is generally more important in Graphtables than spreadsheets. One possible solution is to display the current sorting order in a small vertical table, allowing the user to change the sorting precedence and direction by dragging the column names.

This visualization assumes all paths begin at edges, but it is conceivable for a user to desire the paths to be used for the first column. This could be trivially supported, but is not expected to be implemented due to the fact that a user could simply sort by a field in the first edge column and then look left and right for the answer.

4.7. Breaking the Spreadsheet Metaphor

While Graphtables holds closely to the spreadsheet metaphor, there are differences that should be pointed out. It doesn't make sense to re-order high level columns (Node1, Edge1, Node2,…) since they only represent the overall type in the display and are numbered and labelled solely for user identification.

An additional difference is that the number of rows is static. Other than the user visually hiding values; the actual number of rows cannot change in the current implementation since the number is tied to the number of unique paths in the graph.

Sub-columns under a main column header can be re-ordered, but only within the main column they are displayed.

Since nulls are ranked differently and are always considered of lower rank than all other values, graphs that contain nulls have the oddity that the list in ascending order is not the inverse of the list in descending order; see Figure 15 and Figure 16 sorted by the Valid field.

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value	Valid ▼	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T	AD	2	D	9						
A	3	T	AC	10	C	2	F					
A	3	T										
C	2	F	AC	10	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
C	2	F										
B		F										
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F
D	9											

Figure 15 Descending column with nulls.

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value	Valid ▲	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
C	2	F	AC	10	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
C	2	F										
B		F										
A	3	T	AD	2	D	9						
A	3	T	AC	10	C	2	F					
A	3	T										
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F
D	9											

Figure 16 Ascending column with nulls.

4.8. Visual Opportunities

While the figures above show a relatively simple visual metaphor, all the techniques that have been used in the past can still be brought to bear. Nodes and edges could also be rendered differently (shape, color, stroke, etc…) based on values in their attributes.

One could imagine rendering charts, icons, images, and maps as the values in sub columns. An interesting blend would be to use the common visual metaphors as seen in Bertin matrices, where the user can see the relative magnitude of values, resulting in the ability to scan quicker, see Figure 17.

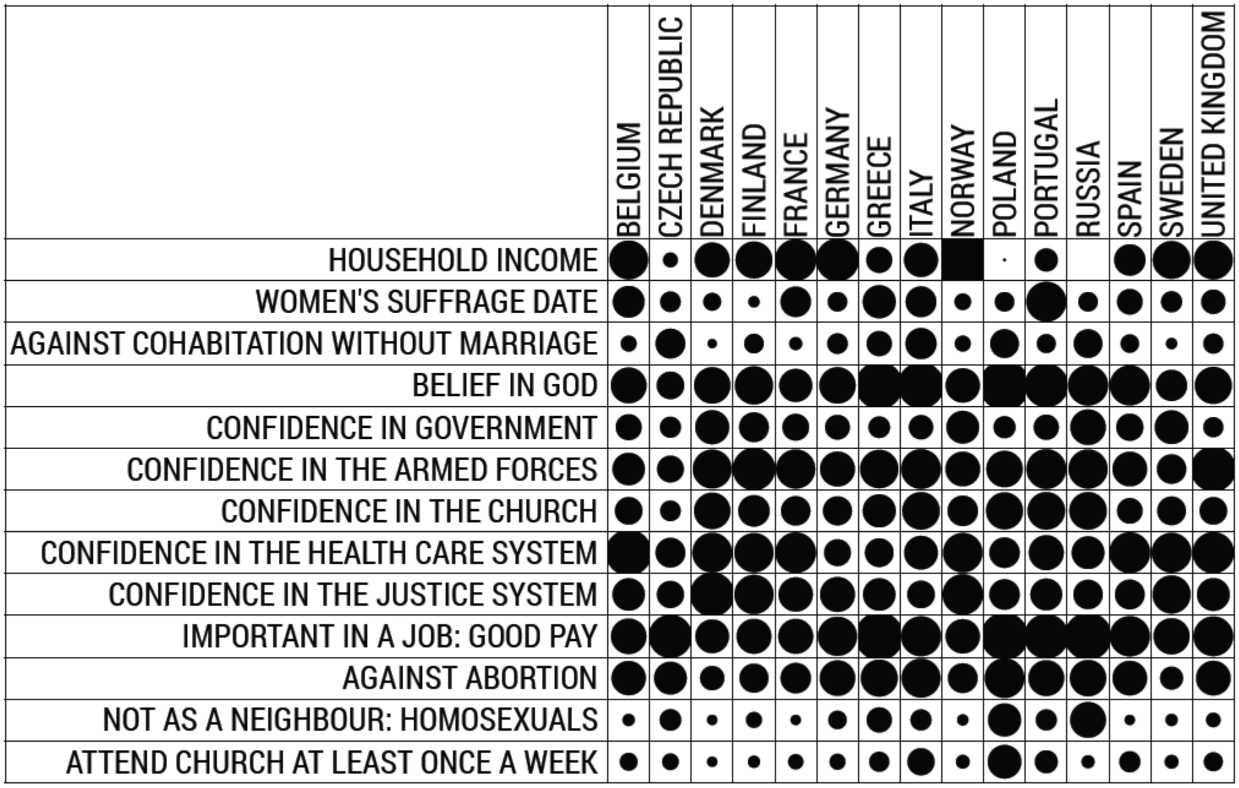


Figure 17 Bertin matrix example. (Perin, Dragicevic, & Fekete, 2014)

Numerous other interaction opportunities exist including all the common brushing and linking operations as introduced by Ben Schneiderman; hovering over one cell could highlight all the other cells that represent the exact same node or edge. The user may wish to preserve the relative order of all sub-columns consistently (Label, Value, Valid as seen through the examples above) so moving the relative position of a sub-column under any main column would change the displayed order for all the sub-columns.

4.9. Initial Java Implementation

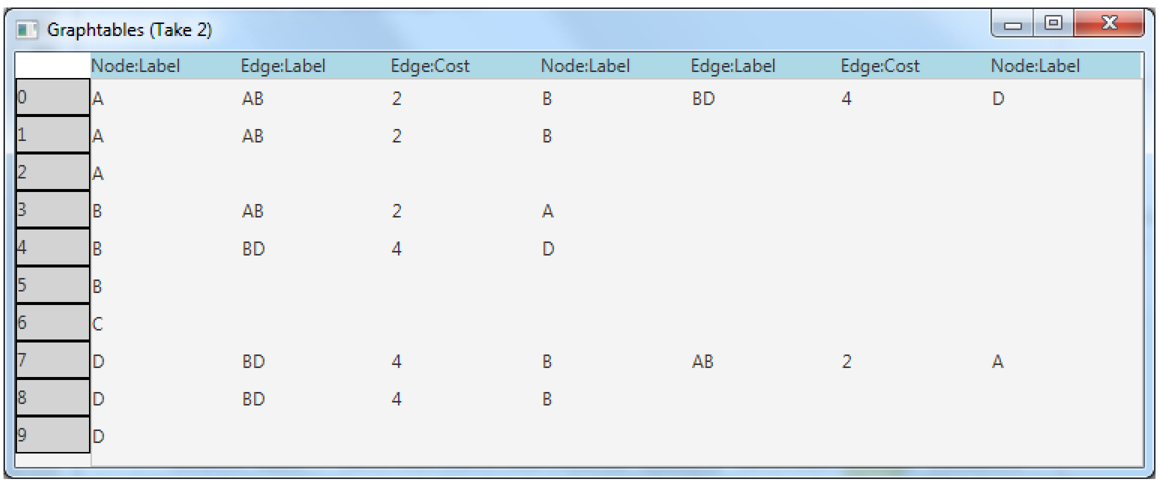
A basic implementation was performed under this task to understand some of the issues that will be present in a full implementation. A major issue was caused by Java's javafx.scene.control.TableView implementation which prevented graphs of more than 500 rows and columns to be interactively panned. In order to address this performance limitation, a prototype replacement for table was developed to provide for out-of-core and on-demand creation of the visual elements when cells are exposed. Out-of-core techniques limit the amount of information kept in main memory and defer loading all the data until it is essential to present or use. These techniques usually minimize the amount of memory, bandwidth, or computation that a system needs to use at the expense of not having all the information immediately available which causes the user to wait until the data can be retrieved from long term storage. This is how applications like Google Earth© allow the user to pan massive geospatial datasets interactively. In the case of Graphtables, the computer had ample memory to store all the information but Java, in particular the JavaFX graphics layer, builds a complete image of the table to make panning more responsive; in this case those "optimizations" became detriments as the vast

majority of the table was not actually exposed and panning all the hidden components in the table was too computationally intensive.

The table implementation provided by JavaFX also failed to provide a rich method for addressing more elaborate column reordering logic. Particularly, for novice users exploring the interface, the system needs to promote learning the interface concepts, and simply rejecting bad positions isn't sufficient; instead, the sub-column will not move further than allowed. Recall that sub columns are limited to being reordered only within the main column for which they occur.

Another issue is in sorting elements of a graph that have heterogeneous attributes. The default Java sorting routines consider null as always higher rank than any value. This means that in sorted sub-columns, the empty values always appeared first. This behavior was opposite of what was desired and a custom sort was implemented.

Figure 18 is a screenshot from the custom table implementation with a graph that has four nodes and two edges. Many features described earlier are missing, in particularly the visual metaphors described above are not implemented. In this case, the user had selected the first column for sorting.



	Node:Label	Edge:Label	Edge:Cost	Node:Label	Edge:Label	Edge:Cost	Node:Label
0	A	AB	2	B	BD	4	D
1	A	AB	2	B			
2	A						
3	B	AB	2	A			
4	B	BD	4	D			
5	B						
6	C						
7	D	BD	4	B	AB	2	A
8	D	BD	4	B			
9	D						

Figure 18 Initial custom table implementation.

5.0 Conclusions

Graphtables provides a simple, intuitive, and powerful structure for directly answering graph specific questions. The promise of such a capability is sufficient to warrant a complete implementation and an experiment to validate the presumption that this method will increase human performance. There are significant hurdles in scaling this metaphor to very large graphs as the system first needs to build a complete set of all possible paths. While this can be done in parallel, sorting the paths and computing user generated fields could pose a significant computational burden. Additionally, there is a question about how long of a path will be useful to the user, but this will only be gleaned from utilizing the capability with real-world scenarios. If it is determined that paths longer than some length offer little benefit, then this optimization could be easily added. It is not necessarily envisioned that this display is meant to completely replace current graph representations, but rather to be the primary mechanism

used to answer graph specific task. In particular, Graphtables does not intend to provide overview. Graphtables offers a huge opportunity and is the only spreadsheet/path approach directly designed to answer graph specific questions.

6.0 References

(MOMA), M. o. (2012). *MOMA | Inventing Abstraction*. Retrieved Dec 2, 2014, from Museum of Modern Art: http://www.moma.org/interactives/exhibitions/2012/inventingabstraction/?page=connections

Gibson, H., Faith, J., & Vickers, P. (2012). A survey of two dimensional graph layout techniques for information visualization. *Information Visualization.* SAGE Publications.

Holten, D., & Van Wijk, J. J. (2009). Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum, 28*(3), 983-990.

Lee, B., Plaisant, C., Parr, C., Fekete, J.-D., & Henry, N. (2006). Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on Beyond time and errors: novel evaluation methods for information visualization* (pp. 1--5). ACM.

Nature America Inc. (2012, February). Networks. *Nature Methods, 9*(2), p. 115.

Perin, C., Dragicevic, P., & Fekete, J. (2014). Revisiting bertin matrices: New interactions for crafting tabular visualizations. IEEE.

Appendix A – Patent Application

Air Force Invention RL10043

METHOD AND APPARATUS FOR REMOVING REDUNDANT INFORMATION FROM DIGITAL DOCUMENTS

STATEMENT OF GOVERNMENT INTEREST

The invention described herein may be manufactured and used by or for the Government for governmental purposes without the payment of any royalty thereon.

PRIORITY CLAIM UNDER 35 U.S.C. §119(e)

This patent application claims the priority benefit of the filing date of a provisional application, serial number 62/087,289, filed in the United States Patent and Trademark Office on Dec 15, 2014.

BACKGROUND OF THE INVENTION

Mathematical graphs which are made up of nodes and edges are pervasive in day to day life. Graphs are even more essential for analysts that rely on graph based data for analyzing domains such as social networks, computer networks, road networks, subway maps and command and control structures. This makes graph visualization and understanding pivotal to effectively using these potentially large and complex graph based data sources.

Traditional graph visualization uses one or more graph layout algorithms to draw rectangles and lines to depict nodes and edges in the graph. These visualizations often rely on algorithms that attempt to layout the graph using poorly balanced aesthetic principles. While the readability of the graphs is the principle purpose of these layout algorithms, increasing graph size and complexity are reducing the effectiveness of these algorithms to allow the user to quickly and easily digest both the structure and the content

of these graphs. This problem is further exacerbated for graphs where the number of nodes greatly exceeds the display area.

Traditional graph visualizations also often fail to maintain the gestalt principle of proximity where the viewer automatically correlates graph elements' proximity to some form of relationship between those elements. Another failing of traditional graph visualizations is that they are ill-suited to address rapid sequential questions where each layout that optimizes a particular question can often cause the entire display to change radically. A layout optimizing a single path through a graph may omit values at the nodes; another layout that bundles edges to give the overall flow within a graph makes it impossible to see which paths actually exist. Overall, each traditional layout compromises which aspects of a graph is displayed.

OBJECTS AND SUMMARY OF THE INVENTION

One object of the present invention is to provide a method and apparatus for displaying graphs as a series of paths in a tabular form.

Another object of the present invention is to provide a method and apparatus for prioritizing the order of the paths that are displayed in order to satisfy dynamic user queries for information.

Yet another object of the present invention is to provide a method and apparatus to allow the user to interactively generate new information based on data both internal and external to the graph while still being able to use these new results for prioritizing the sorting order of the paths.

The invention disclosed herein provides a method and apparatus for displaying graphs as a series of paths with the ability to filter and sort those paths based on intrinsic and extrinsic values. In particular, this invention allows the user to display and intuitively interact with graphs using a common spreadsheet style metaphor. A graph consists of a set of edges and nodes where edges connect nodes to nodes. Each edge and/or node can have any amount of other data associated with it, whether integer, real, boolean, textual or otherwise. The present embodiment of Graphtables allows the user to interactively: configure the sorting order of the columns of the table; define which data to display in each column; and compute new values or fields. By providing these few, but

powerful set of operations, the user can quickly get a list of paths through a graph to answer targeted questions such as: which paths are of length 5 starting from a particular node; which nodes are directly connected to a particularly node; and display all the paths that have a node with an address that is also 5 minutes away from another user-supplied address.

According to an embodiment of the present invention, Graphtables, comprises the steps of: accepting an input graph from the user; computing all possible paths through the graph such that each path is unique and has no cycles; and display each path in a single row where upon initialization that the first row starts with the shortest paths and the last row contains the longest path.

According to the preferred embodiment of the present invention, Graphtables, the spreadsheet metaphor is slightly changed in that the user is presented with a set of overall columns in the order of Node, Edge, Node, Edge, etc. There is one column for each element in the longest path, so if the longest path is of length 15, there are 15 main columns each labelled with Node or Edge. In the preferred embodiment, each main column is broken up into sub columns where each attribute of the node or edge is displayed in a sub column. The number of sub columns is equivalent to the number of unique attributes for all nodes or edges. This allows the present invention to accommodate nodes or edges with non-homogenous data and data types. In the preferred embodiment, when a node or edge does not contain a particular attribute, it simply doesn’t display any value in that cell. Other embodiments may choose to display NULL or other value or symbol.

According to a feature of the present invention, Graphtables, the user can re-order the sub columns not labelled Node or Edge. In the preferred embodiment, this only alters the location within that one Node or Edge but in another embodiment, changing the column display order for sub columns could alter the display order for the other Node or Edge columns. There is no utility in moving the main columns labelled Node and Edge as they only let the user know that the column is displaying node or edge information.

According to a feature of the present invention, Graphtables, the user can sort any number of columns as is common in spreadsheet applications. The user selecting columns to sort is equivalent to a complex graph matching search where more relevant

results are displayed first, but Graphtables achieves this effect without requiring any complex textual input. It is not necessary that each node or edge contain the same number of attribute data values. The preferred embodiment allows the user to decide when a node or edge does not contain a value, whether to consider that the node or edge lacking a value is displayed earlier or later in the table.

According to another embodiment of the present invention, Graphtables, the user can filter the data by all or any of the following: 1) all nodes and edges, 2) any subset of nodes or edges, and 3) any number of rows. When a path fails the filter, the preferred embodiment does not hide that path, it just makes it sorting order to be later in the list. This allows the user to still sort and view the filtered results in context with the other information. The preferred embodiment still sorts the filtered data in the same way as specified by the user and displays a line to depict that the rows below that line are filtered. This embodiment effectively converts filtering to a simple ranking calculation, showing possible relevant results where other graph implementations would have excluded those nodes or edges from the graph entirely. Those traditional implementations do not allow the user to see that there are paths that might have closely matched their filter.

According to another feature of the present invention, Graphtables, the user can insert main columns or sub columns that generate derived data for the entire graph, or any number of steps along the path. If the inserted columns are at the main level, the inserted columns are moveable next to any other already existing main column. If the inserted column is a sub column, that sub column location can be moved but is limited to the inserted main column.

According to another feature of the present invention, Graphtables, the user can choose to bundle edges so that if two neighboring rows share the same edge, the node is only displayed once and the edge is only displayed once on the first occurrence and all subsequent contiguous edges are displayed as edges from the primary edge.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 depicts a sample graph with the default sorting, called the Initial State.

FIGURE 2 shows the result of sorting by a single column from the Initial State.

FIGURE 3 shows the result of sub sorting by a second column from the state depicted in Figure 2.

FIGURE 4 shows the result of sub sorting by a third column from the state depicted in Figure 3.

FIGURE 5 shows the result of filtering the entire table from the Initial State.

FIGURE 6 shows the result of filtering a single column from the Initial State.

FIGURE 7 shows the result of filtering a single column from the state displayed in Figure 3.

FIGURE 8 shows the result of inserting a custom main level computed column.

FIGURE 9 shows the result of sorting a different column from the Initial State.

FIGURE 10 shows the result of edge bundling from the state displayed in Figure 9.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

This invention displays a user defined ordered set of paths with optional filters and user defined columns. In particular, this invention provides an interface to explore graph paths that satisfy extrinsic user needs.

Referring to **FIGURE 1**, the graph G is displayed in its initial state assuming the user had loaded a graph with the nodes and edges as defined in **100**. **120** displays the same graph as the traditional node-link style of the same graph as defined in **100.** **140** contains the additional metadata for each node and edge. **160** is the canonical depiction of the same graph as defined in **100, 120,** and **140**. **160** consists of 5 major columns, where each major column is named based upon the type of item that it contains, either nodes or

edges and has a monotonically increasing column number for each node-edge pair and has subcolums as in **165**. The number of columns in **160** is defined based on the length of the longest path through the graph as defined by **100.** The titles of the subcolumns as in **165** are based on the metadata in the underlying graph as defined by **140**. Each row of the table as in **170** displays a single walk through the graph and its contents are based on the sorting order of the columns in **165**. The canonical representation for a node as in **175** is to simply surround the contents of the primary column as in **160** with a rounded rectangle. The visual depiction in **175** is solely to improve the user's interpretation and in no way affects the ability to: sort contents, alter the order of the columns, or insert user defined columns. The canonical representation for an edge as in **185** is to simply to draw a thick line behind the contents of the primary column as in **160**. The visual depiction in **185** is solely to improve the user's interpretation and in no way affects the ability to: sort contents, alter the order of the columns, or insert user defined columns. **180** displays the canonical method of displaying NULL values metadata fields as empty cells.

Referring to **FIGURE 2** is the result of sorting the *value* column (**200**) in the graph as defined by **140** in descending order. This action causes the rows to be reordered based on the contents of that cell. Note that the entire path stays together and entire rows are sorted, not just the contents of a single column or subcolumn.

Referring to **FIGURE 3** is the result of sorting the *weight* column (**300**) in the graph as defined by **140** in ascending order after previously sorting was applied as in Figure 2. Note that sorting precedence is applied, and the sort as applied in Figure 2 stays consistent.

Referring to **FIGURE 4** is the result of applying a graph filter **(400)** based off the initial state as displayed in Figure 1. A graph based filter evaluates all elements in all paths for the entire graph. If any element fails the check, then the entire path fails the check and is considered filtered. In the canonical implementation, a filter doesn't eliminate the element, but instead serves as a sorting order modifier. In the canonical implementation, all non-filtered elements are displayed before all filtered elements, but regardless of filtered or non-filtered, all elements are still sorted. In the current embodiment, a dark line is drawn (**450**) to visual separate the filtered from non-filtered elements.

Referring to **FIGURE 5** is the result of applying a column based filter (**500**) off the initial state as displayed in Figure 1. A column based filter only compares the filter equation versus elements in the column in which it is defined. If any element in the subordinate column (**500**) fails the filter evaluation then the entire row (aka path) fails the check and is considered filtered. In the canonical implementation, a filter doesn't eliminate the element, but instead serves as a sorting order modifier. In the canonical implementation, all non-filtered elements are displayed before all filtered elements, but regardless of filtered or non-filtered, all elements are still sorted.

Referring to **FIGURE 6** is the result of sorting the *label* column (5**00**) in the graph as defined by **140** in descending order. This action causes the rows to be reordered based on the contents of that cell. Note that the entire path stays together and entire rows are sorted, not just the contents of a single column or subcolumn. Additional to the sorting, **FIGURE 6** depicts the process of edge bundling. Subsequent rows that would duplicate the same value (**520, 540**) are displayed as a forked edge (**560**) instead of duplicating the values. This reduces the visual clutter and is intended to preserve the visual the uniqueness of each path.

Referring to **FIGURE 7** is the result of inserting a custom top level column (**700**) used to compute some value based on the contents of the path or the graph. In this case, the value as seen in **750** displays the computed path length. These computed column(s) can also be used as sorting column and work the same as in previous examples. Unlike the path based main level columns (**160**), custom columns can be relocated to anywhere the user desires. Additionally, subordinate custom column headers

Other embodiments do not display filtered elements, and the number of failed filters can also be used to further reduce the rank of a filtered path. This means that a path that only fails one filter would rank higher than a path that failed two filters.

Other embodiments may choose to hide columns or rows to reduce the amount of displayed content. Other embodiments may also dramatically change the visual metaphor of using rounded rectangles for nodes and lines for edges.

While the preferred embodiments have been described and illustrated, it should be understood that various substitutions, equivalents, adaptations and modifications of the invention may be made thereto by those skilled in the art without departing from the

spirit and scope of the invention. Accordingly, it is to be understood that the present invention has been described by way of illustration and not limitation.

What is claimed is:

1. Method for displaying graphs as a series of paths, comprising the steps of:

creating a series of paths through the graph;

displaying the series of paths using a tabular based metaphor;

sorting the paths based on user supplied criteria;

inserting user defined columns to augment the data; and

filtering the data based on user supplied criteria

2. Method of claim 1, wherein said step of displaying the paths comprises the steps of:

ingest provided graph information;

generate all applicable paths through the graph; and

display each path as a series of node-edge-node-edge-…columns in a tabular row configuration where:

each piece of metadata is displayed in a labelled sub column;

users are allowed to select to sort by this column in ascending or descending order;

user can reorder the visual placement of these sub columns such that it doesn't move outside its parent column; and

disallow the user to alter the order of node-edge-node-edge… columns as this operation doesn't make sense.

3. Method of claim 2, wherein said step of sorting the paths comprises the steps of:

user or application driven sorting precedence of:

sorting all paths in ascending or descending order by the first column identified using the natural ordination or user supplied ordination;

sorting all paths in ascending or descending order by subsequent column(s), in the order as defined by the user, using the natural ordination or user supplied ordination such that the path rank of previous orderings is still observed.

4. Method of claim 3, wherein said step of filtering the paths comprises the steps of:

user defining a filter whether at the graph, primary node or edge column or sub column filter:

user defines acceptance criteria for path or sub path filtering;

paths that fail the filtering criteria are displayed after those that pass the filtering criteria;

an optional visual marker is displayed separating the paths that pass and those that fail the criteria; and

sorting as in claim 1 is still observed for the filtered path(s).

5. Method of claim 4, wherein said steps of augmenting path information by inserting user defined columns comprises the steps of:
user inserting a column as a sibling to the main node or edge columns or as a subcolumn of a particular node or edge:

user defines the name of the column;

user chooses predefined algorithms or defines a new process for providing new detail as a string, number, or other computable value capable of being sorted;

Computed values are displayed in line with the path in which they calculate and then can be used for sorting as in Claim 1.

Computed columns at the main level can be moved to any position to interleave with nodes or edges.

Computed columns at the lower level can only be moved within that node or edge.

6. Method of claim 5, wherein said steps of bundling edges comprise the steps of:
IF the node in the row displayed above this current node or edge contains the same information, THEN

do not draw the content and perform the following:

IF the node traverses the same edge, THEN

do not display the edge in the normal fashion, and instead show it as a fork off the previous displayed edge.

OTHERWISE

display the edge in the traditional method

OTHERWISE

display the node in the traditional method.

ABSTRACT OF THE DISCLOSURE

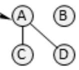
Method and apparatus for displaying and identifying relevant paths through a graph by displaying them in a tabular format and providing user defined and computed values and filters. Each graph is represented as a set of nodes and edges. Each paths through the graph is displayed as a row in a table where the user can apply sorting, filtering, and compute intrinsic or extrinsic information to augment the data. It is proposed that the user performs a useful set of these operations to identify paths through the graph to identify relevant traversals to gain insight of the graph in general or to answer specific questions. Unlike traditional graph visualizations, this mechanism displays in rank order, all paths that could be useful in answering the questions. The invention presents a fundamentally new way for structured navigation, inspection, and augmentation of graphs using paths.

Appendix B – Graphics for Patent Application

100

G(v,e) = ({A,B,C,D}, {{A,C}, {A,D}})

120



140

Nodes	Label	Value	Valid
A	A	3	T
B	B	NULL	F
C	C	2	F
D	D	9	NULL

Edges	Label	Weight
AC	AC	10
AD	AD	2

160

165

170

175

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T										
B		F										
C	2	F										
D	9											
A	3	T	AC	10	C	2	F					
C	2	F	AC	10	A	3	T					
A	3	T	AD	2	D	9						
D	9		AD	2	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
D	9		AD	2	A	3	T	AC	10	C	2	F

180

185

Figure 1: Initialization State

200

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value ▼	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
D	9											
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F
A	3	T										
A	3	T	AC	10	C	2	F					
A	3	T	AD	2	D	9						
C	2	F										
C	2	F	AC	10	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
B		F										

Figure 2: Sort State 1

300

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value ▼	Valid	Label	Weight ▲	Label	Value	Valid	Label	Weight	Label	Value	Valid
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F
D	9											
A	3	T	AD	2	D	9						
A	3	T	AC	10	C	2	F					
A	3	T										
C	2	F	AC	10	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
C	2	F										
B		F										

Figure 3: Sort State 2

400

450

Filter| Value < 5												
Node 1			Edge 1		Node 2			Edge 2		Node 3		
Label	Value	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T										
C	2	F										
A	3	T	AC	10	C	2	F					
C	2	F	AC	10	A	3	T					
B		F										
D	9											
A	3	T	AD	2	D	9						
D	9		AD	2	A	3	T					
C	2	F	AC	10	A	3	T	AD	2	D	9	
D	9		AD	2	A	3	T	AC	10	C	2	F

Figure 4: Filter from Initial State

500

Node 1			Edge 1		Node 2			Edge 2		Node 3		
Filter| Value < 5												
Label	Value	Valid	Label	Weight	Label	Value	Valid	Label	Weight	Label	Value	Valid
A	3	T										
C	2	F										
A	3	T	AC	10	C	2	F					
C	2	F	AC	10	A	3	T					
A	3	T	AD	2	D	9						
C	2	F	AC	10	A	3	T	AD	2	D	9	
B		F										
D	9											
D	9		AD	2	A	3	T					
D	9		AD	2	A	3	T	AC	10	C	2	F

Figure 5: Column Filter from Initialization State

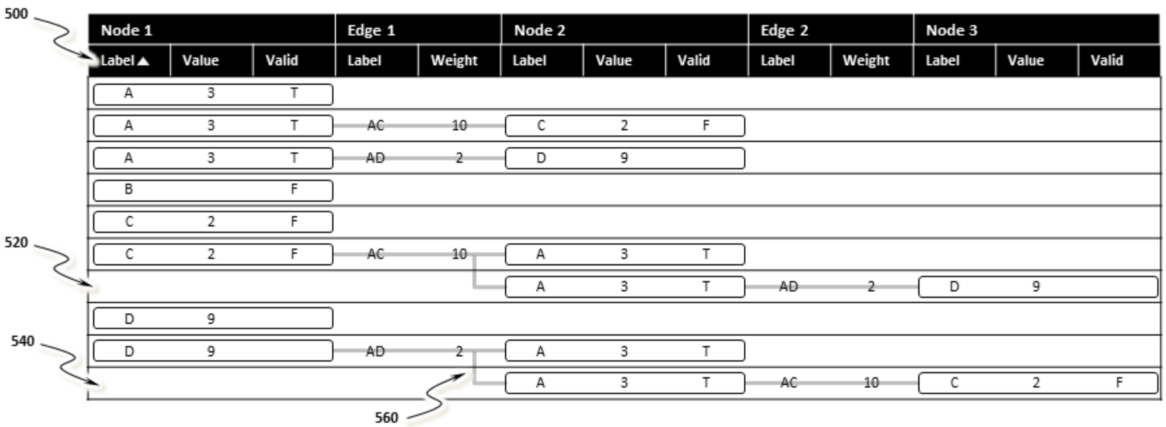


Figure 6: Sort by Label then Bundle Edges

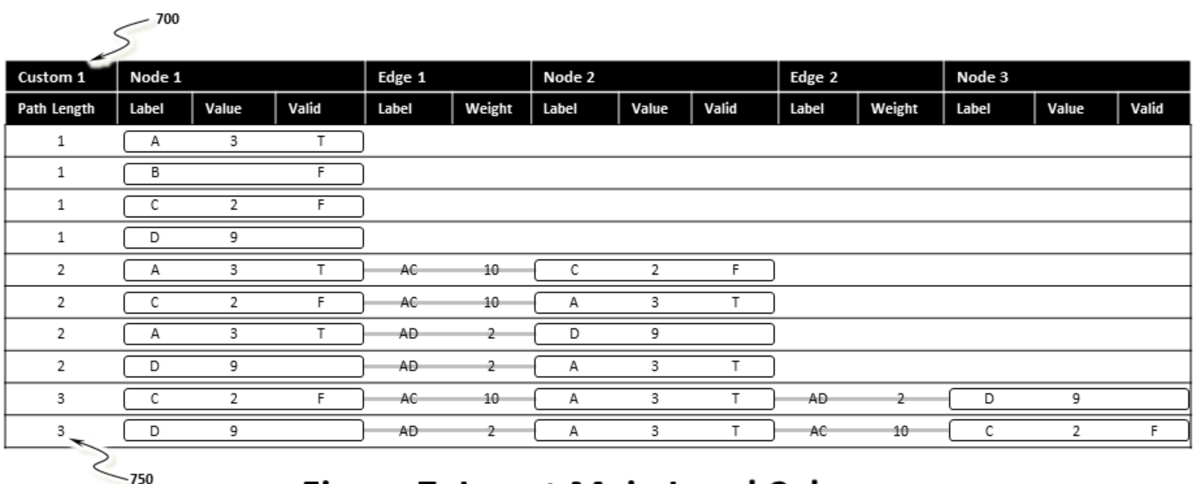


Figure 7: Insert Main Level Column

List of Symbols, Abbreviations, and Acronyms

C2 …………………………………………………. Command and Control

DOD ……………………………………………….. Department of Defense